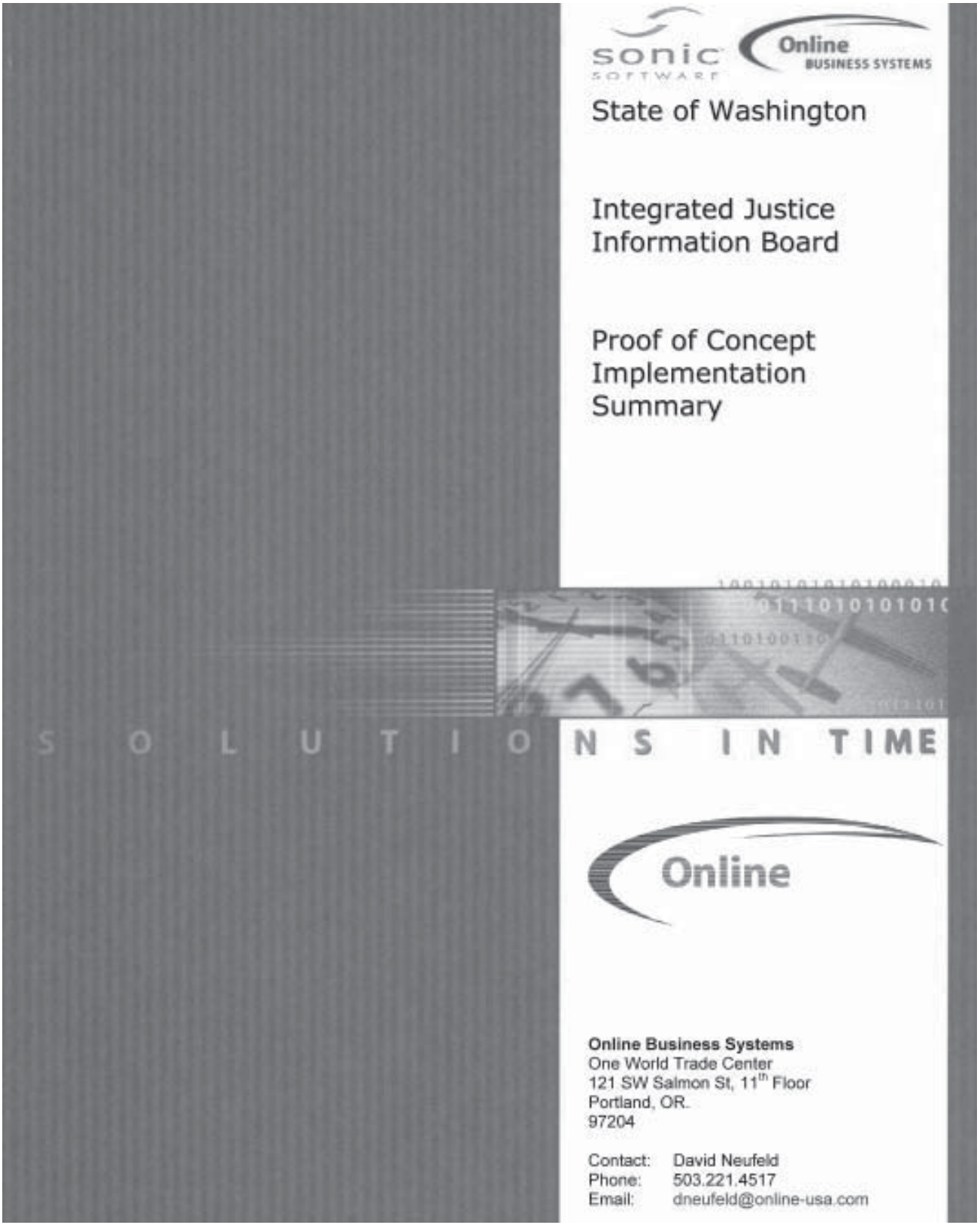

sonic
SOFTWARE
Online
BUSINESS SYSTEMS
State of Washington
Integrated Justice
Information Board
Proof of Concept
Implementation
Summary
SOLUTIONS IN TIME
Online
Online Business Systems
One World Trade Center
121 SW Salmon St, 11th Floor
Portland, OR.
97204
Contact: David Neufeld
Phone: 503.221.4517
Email: dneufeld@online-usa.com

1. Business Summary

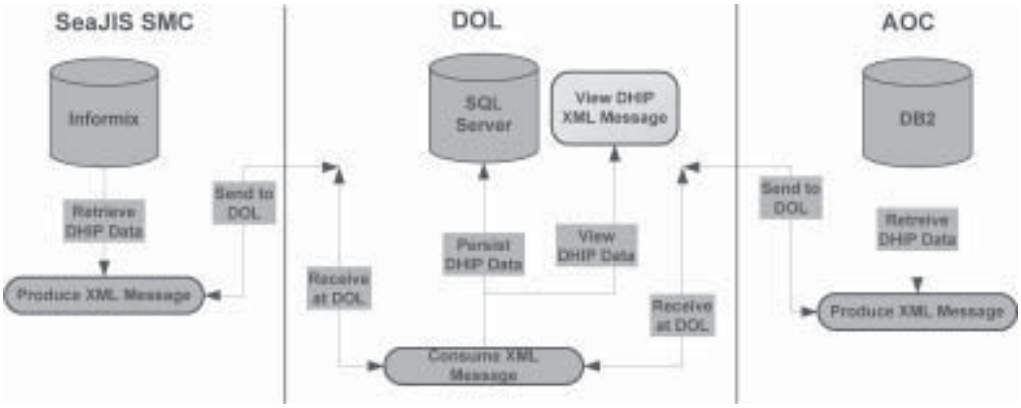
Online Business Systems (Online), together with our partner Sonic Software (Sonic) and on behalf of the State's Justice Information Network (JIN), has developed a proof of concept (PoC) message exchange of Driver History Initiative Project (DHIP) court event data between the City of Seattle's Municipal Court (SMC), and the State's Administrative Office of the Courts (AOC) and Department of Licensing (DOL).

We understand the primary objective of the proof of concept is to allow JIN to explore the viability of distributed integration architecture. As such, Online developed a **service-oriented integration infrastructure** based on Sonic's ESB technology platform. Online believes that such a technology platform for capturing, processing, and securely sharing mission-critical justice information where and when it is needed is the foundation upon which the State of Washington's JIN should base both near and long term integration initiatives.

"Combining message oriented middleware, Web services, transformation and routing intelligence these high-function, low-cost ESBs are well suited to be the backbone for service-oriented architectures and the enterprise nervous system."
Roy Schulte, Vice President and Research Fellow - Gartner, Inc.

Issues related to the interoperability of systems and languages were also addressed by the PoC. Interestingly, of the agencies involved in the PoC, the DOL would consider itself to be a "Microsoft shop" while the AOC a "Java shop". The PoC itself was comprised of service components from both camps. The Sonic ESB technology solution is as open an integration technology as exists in the marketplace today and perfectly suited to the heterogeneous environment of the State's JIN community. It is important to note that the Sonic ESB architecture is designed to run within an organization's existing network / communication and security infrastructure including existing server hardware and operating systems. By leveraging an organizations existing infrastructure, Sonic can dramatically reduce the cost of deployment and accelerate the development and implementation cycle.

The diagram below represents the planned workflow scope of the JIN PoC. The diagram depicts XML messages containing DHIP information being rendered from information persisted in the SMC's MCIS/Informix database and the AOC's DB2 database. The XML messages are in turn transmitted from the SMC and the AOC to the DOL via Sonic's ESB technology platform across the IGN. In order to avoid any lasting or invasive changes to the DOL's existing production systems related to drivers licensing and vehicle titling and registration, a small application was developed by the Online team representing the receiving system at the DOL. The application was developed in MS .NET technology and allows incoming messages to be viewed.

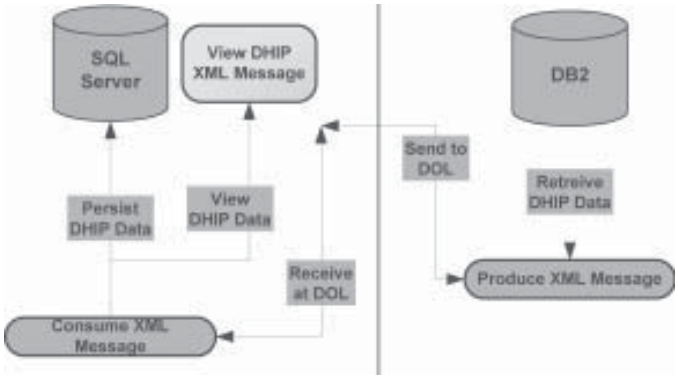


In early June, the Sonic ESB and related integration software was installed at the AOC and DOL. The integration components developed by Online were implemented and deployed. The exchange of DHIP data in XML formatted messages between these two agencies was successfully demonstrated and discussed. The report from the JIN program director dated June 10, 2004 read as follows:

The second JIN proof of concept, an exchange of failure-to-appear information among Seattle Municipal Court,
AOC, and DOL, is now underway. We established connectivity between AOC and DOL on Thursday June 10. The
process went smoothly, and it is interesting to note that DIS involvement in the exercise was not required.

Unfortunately, the SMC was unable to participate in the PoC exchange due to scheduling constraints. The SMC had (has) established a “freeze” on any additional changes and/or demands on its IT services due to the initial implementation of the SeaJIS ESB message exchanges (June – July) and a planned server migration (July). In addition to the scheduling challenges of the SMC, the DOL had a large URA implementation scheduled for the July long weekend that presented a scheduling challenge to the JIN PoC. Shortly after the initial implementation of the message exchange between the DOL and AOC, the DOL asked to reclaim the server that had been loaned to the integration exercise. The diagram below represents the actual implementation of message exchanges between the AOC and DOL only.

The State awarded the Online team the opportunity to participate in one of two JIN sponsored PoC exercises in March 2004. It was decided at that time to have Online deliver its PoC subsequent to the first exercise. Basic refinements to the statement of work were dealt with thru April and early May. In late May a member of Online’s delivery team from the SeaJIS project began work on developing the components required for the JIN PoC. On June 9th-10th the ESB-platformed technology components were installed at the DOL and AOC and DHIP XML messages were successfully exchanged between the AOC and the DOL.



The actual work effort required to build, install, implement, deploy, test and demonstrate the PoC message exchanges was approximately 160 hrs. This estimate does not including the delivery and account management activities between the JIN and Online.

2. Technical Summary of Work

As a general statement, the data integration processes developed for the JIN PoC represent a Service Oriented Architecture (SOA). The concept of a SOA is to develop components of discreet functionality or business logic known as services. The collection and interaction of these discrete services comprise the SOA.

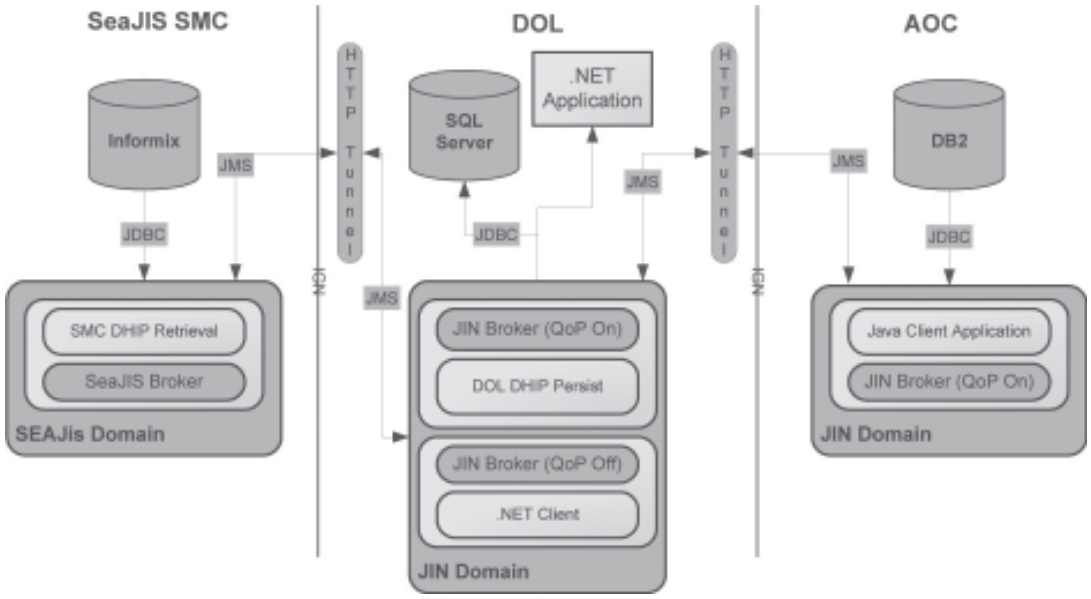
Each individual service is responsible for a clearly defined business task. In general, services should have the following design characteristics:

- Loosely coupled. As each service is only responsible for its particular task, it does not know about or care how it is used to form a complete data integration process. This allows services to be decoupled from one another and increases the reusability of the service in many data process integrations.
- Coarse grained. Service interfaces are designed to more closely model real world business processes, not the low level interfaces of the programming language used to construct the functionality of the service.
- Asynchronous communication. Services may act as sender, receivers or both. With an asynchronous model a particular service is not dependent on another service being available for the service to perform its functionality.

The following components were used to provide the platform by which the individual services were bound together into the JIN PoC SOA:

Service Bus	A service bus provides the mechanism for individual services to reliably communicate with one another. Sonic Enterprise Service Bus (ESB) is used as the service bus. The underlying communication mechanism used by the JIN PoC is SonicMQ, which uses Java Messaging Service (JMS) over TCP/IP protocol.
Sonic ESB Services	Sonic ESB services will provide the backbone for services developed for the JIN PoC. To cut down on the number of service types used by the JIN application, a pattern was used where no business logic was performed in the actual Sonic ESB service. Business logic was abstracted out of the Sonic ESB service and encapsulated into an action. In essence, the Sonic ESB service is used as a façade to provide connectivity to the Sonic ESB backbone for messaging, configuration information and execution of service methods in the correct order by the Sonic ESB engine.
Action	An action contains the actual business logic that will be performed. If a particular business process is general enough, it is possible that a particular action be reused by many Sonic ESB services. Configuration parameters passed into the Sonic ESB service will allow the action to be configured dynamically, allowing for reuse of the action. The current design pattern is for a Sonic ESB service to use one action at a time. To assist the action in providing business functionality, Data Access Objects (DAO) and XML Serializers were used. A **DAO** provides the functionality to connect to a database to perform one of the following types of data processes: o Persist. Information is persisted into the database. A persist can take the form of an insert, an update or a delete. o Retrieve. Information is retrieved from the database. In general, the design pattern of the DAO is to act upon one table in a particular database. If information is needed from multiple tables, an action would either use multiple DAOs or a DAO would be constructed that contained the logic to access more than one table depending on the business processing required. A DAO is not designed to work across multiple databases. If an action were to require this functionality, multiple DAOs would be used, chained with multiple action instances. The DAO make use of two Java technologies: Java Database Connectivity (JDBC) and Java Architecture for XML Binding (JAXB). JDBC provides programmatic access to relational data from the Java programming language. Using the JDBC API, applications written in the Java programming language can execute SQL statements, retrieve results, and propagate changes back to a database. JAXB provides an API and tools that automate the mapping between XML documents and Java objects. JAXB compiles an XML schema into one or more Java technology classes. The combination of the schema-derived classes and the binding framework enable developers to perform the following operations on an XML document: o Marshal the Java representation of the XML content into XML content o Unmarshal XML content into a Java representation o Access, update and validate the Java representation against XML schema constraints An action uses an **XML Serializer** to transform a business object to XML or to transform XML to a business object. The XML Serializer makes use of JAXB to perform the various types of transformations.

The following diagram represents the primary technical components that comprised the message exchange proof of concept described in the Business Summary above.



Department of Licensing
In the Department of Licensing domain, two Sonic 'brokers' were established. Brokers are the management and coordination entities in the bus – representing the messaging system. The primary or secured broker (QoP On) acts the coordinator between departments. It locally manages the DOL queues while ensuring the use of high power cipher suites like AES-256. This secured broker is the one that was exposed to the Internet or the GDN and as such reduced the possibility of a hack to a computational impossibility. Additionally in the primary broker was a custom written service. This service was responsible for reading an incoming message from a negotiated source, to a SQL Server database.

The second broker (QoP Off) was established for internal use. It is convenient to use a separate broker for additional management information, but in this case, it was required to support a .NET client. Unfortunately, the .NET client framework is not capable of using AES-256 yet. The second broker is used to authenticate against the primary broker securely, while the .NET client authenticates against the second broker without the cipher.

The .NET client is an example of an application that is notified when a message arrives on its queue. This is an asynchronous process, meaning that the application does not burn through CPU cycles asking the question "is there something for me now?" The application is awoken when there is something addressed to it. In this case, a message is simply displayed, but it can become much more complex. It could require a user input, or it could simply persist the message to a database.

Administrative Office of the Courts
At the AOC, a single Sonic broker was established responsible for the secure communication between other departments. The communication could have been in the form of a JMS tunnel, an HTTP POST method or a Web Service. For the AOC, an HTTP-direct connection was used which allowed for secure communication while using an HTTP transport for traffic between proxy servers and firewalls.

Deployed to the AOC broker were two services. The first service was a scheduler service, responsible for initiating the whole process. The second service retrieved data from a DB2 database and generated a Justice XML compatible message. The message was then transmitted to the AOC. This service was a wrapper to the existing application provided by AOC that created a batch XML message of driver history information. The existing application was also

rebuilt, but to transmit the XML as a client using an asynchronous message, as opposed the FTP method used before.

Seattle Municipal Court
For the reasons previously stated, it was agreed to cancel the PoC implementation at the SMC. ESB deployment and services similar to those of the AOC described were developed for the SMC but not implemented.

One service intended to be deployed to the existing SeaJIS domain performed the same task as the AOC service for reading data from a database and creating an XML message. However, this service was developed to use an XML marshalling framework for XML creation and read data from an Informix database. It should be noted that the XML that was produced was of a single row of data, as opposed to the many rows sent at AOC. The single row approach is representative of a real-time, or event-driven deployment. Less real-time was the use of a reused scheduler service that triggered the process, as the SMC database was not capable of creating an event to start the process. This could have been in the form of a Java or .NET trigger, an HTTP post or even a JMS enabled message.

Connectivity
The DOL installation required that the HTTP acceptor accept connections from external agencies. As such, firewalls were configured to accept external traffic to the destination. This will require at least one port. If SSL is a requirement, then an additional port will be required: One to accept traffic over HTTP and another for HTTP over SSL. It would be preferred to accept only connections over SSL, but the risk is in alienating legacy applications that do not support SSL but are web enabled.

SSL provides an encrypted tunnel to the data that is being transmitted. For a full system, both mechanisms should be enabled with a strong cipher suite being required for an SSL connection. AOC and SMC do not require an acceptor because the information is originating at these departments. The use of certificates is also a mechanism that can be used with SSL to ensure proof of identity of the client who is authenticating.

The port 2510 is a recommended HTTP Tunneling port. HTTP Tunneling is a HTTP enabled listener that can work with normal Sonic client connections or with standard HTTP Posts to support legacy applications. DOL must be accepting standard TCP traffic through its firewall on this port.

Data Architecture
In terms of message content, the scope of the JIN PoC was to include information sourced from both the AOC and SMC end-point systems destined for the DOL related to DHIP court events. For the purposes of the JIN PoC we did not develop an intermediate or common business format between the AOC, DOL, and SMC. Instead, message formats were designed around existing DHIP message formats made available to us by the SMC and AOC.

Although the scope of the data architecture work related to the JIN PoC was constrained as above, we recommend a “best practice” JIN data architecture be built around several fundamental concepts.

1. Law Safety Justice (LSJ) industry standards
    (a) The use of Justice XML ensures that the definition of information being sent across interfaces can be reused by all authorized agencies. Because these messages conform to published definitions of LSJ artifacts, the parsing of this data is consistent.
2. Technology Standards
    (a) XML based message ensures that all mainstream technologies in use at any agency will have the capability to interoperate with these messages
    (b) The use of XSL standards to transform messages from their native formats into the intermediate business format ensure developers are re-using existing skills as well as leverage standards based technology.
3. Flexible and Scaleable Data Architecture Design (Loosely coupled interfaces)
    (a) By isolating the message formats from the endpoint systems, changes to these systems will have no impact on the exchanges after they leave the endpoint system. This results in drastically reduced maintenance costs as well as increased responsiveness to business needs as they evolve over time.

3. IMPLEMENTATION LOG

- **Initial DOL Installation Implementation**
  (Wed June 9 8:30a – 11:30a)

  Tasks
  1. Installation and Configuration of Sonic Software (MQ and ESB)
  2. Installation and Deployment of JIN PoC Components
  3. Test simulated message acceptance

  Issue Log
  - o DOL Staff extremely busy with upcoming URA Implementation
    a. DOL resources were understanding and cooperative – but there was more emphasis on tactical implementation ("getting the job done") then in education and awareness of the approach and technology
    b. A new target was set to demonstrate AOC-DOL connectivity by same-day 3pm.
  - o IP addresses provided to and tested by the DIS different from the actual IP addresses – proved to be nonissue.

- **Initial AOC Installation Implementation**
  (Wed June 9 1:00p – 4:30p)

  Tasks
  4. Installation and Configuration of Sonic Software (MQ and ESB)
  5. Installation and Deployment of JIN PoC Components
  6. Establish ESB Connectivity between AOC and DOL
  7. Test simulated message acceptance

  Issue Log
  - o IP addresses provided to and tested by the DIS different from the actual IP addresses. AOC firewall was not configured to allow incoming connection from DOL (required by the PoC to handle message received acknowledgements)
    a. AOC resources responsively were able to reconfigure the firewall to allow incoming DOL traffic
  - o PoC Server provided by AOC was a somewhat overworked box running a number of VMs – the fact that Sonic installed and the integration environment ran under this environment is we believe a real testament to the lightweight and standards-based technology approach
  - o Source dB was initially SQL-Server not DB2 as assumed – the effort related to this change caused us to spend more time in this step and the trickle-down effect was that we were unable to perform the AOC-DOL message exchange demo by the 3pm target.

- **DOL Demonstration (Message Exchange + Receipt + Consumption)**
  (Thu June 10 10:00a – 11:00a)
  Tasks
  8. Demonstration of simulated message exchange of DHIP data sourced from AOC transported from AOC-to-DOL via IGIN/ESB and received/consumed at DOL
  9. DOL specific JIN PoC Education and Awareness discussion (Graham Hainbach from Sonic and Robin Griggs from the DOL)

  Issue Log
  - o Due to issues with DB2 connectivity – simulated DHIP messages were transmitted from the AOC and DOL.
    a. In the context of the PoC and from the perspective of the DOL – the demonstration accomplished the objectives of demonstrating connectivity to the AOC and demonstrating a couple of alternatives as to how messages at the DOL were received and processed by end-point systems

- **AOC Demonstration**
  (Thu June 10 4:15p – 4:45p)

  Tasks
  10. Installation of DB2 Client and Licensing Component
  11. Demonstration of data Extraction from DB2 database – transformation to XML message format – transmission of message exchange from AOC to DOL
  12. AOC specific JIN PoC Education and Awareness discussion

  Issue Log
  - It was resolved to go back to the DB2 source database. The Online implementation team worked a number of angles before resolving the issue around 2:30pm

4. Discussion Points

A number of questions and/or discussion points about the project approach and/or PoC solution architecture as implemented by the Online/Sonic team were raised throughout the proof of concept exercise. In the following sections we highlight a few of these discussion threads.

4.1 On Proof of Concept versus Production Deployment

How does a full-scale (or incremental) deployment differ from the Proof of Concept exercise and how might it proceed.

In regards to how a full-scale deployment might proceed, the activity to begin with is mostly one of collaborative preparation and planning the parameters of the project – including the supporting organizational structure – that would be required to support the business objectives of the JIN community as well as the technical objectives and constraints of the JIN/DIS.

- ***Develop and Conduct Program of Education and Awareness.*** Deliver an education and awareness program to the constituents of JIN, the DIS, and other JIN stakeholders and partners as required. The program would describe in detail the integration technologies and project approach that is being recommended by JIN and their technology partners.

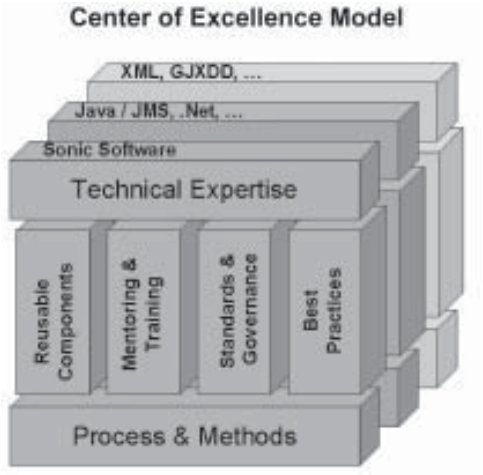
- ***Establish Service-Oriented, Standards-Based, Platform and Language Agnostic Integration Architecture.***
  - ESB Characteristics
    - 100% Standards-Based (JMS, XML, JCA, SOAP, XSLT, UDDI…)
    - Highly Secure (Plug-able Security and Cipher Suites)
    - Messaging Reliability (Guaranteed once-only Delivery)
    - High Performance (DRA Production Proven)
    - Incremental Deploy (Across widely distributed Enterprise)
    - Vast Scalability (Distributed Clustering Architecture)
    - Enterprise Management (Distributed computing, Central Administration and Monitoring, Federated security)
  - ROI-related Business Benefits include:
    - Lower Initial Cost by leveraging existing IT infrastructure assets and reducing the number of platforms needed.
    - Lower Implementation cost through standardization and ease-of-implementation
    - Longer life by using standards-based architecture that provides adaptability

- ***Initial Production Message Exchange Implementation.*** Implement an initial message exchange upon the integration foundation between two or more State agencies or constituents of the State's Justice Information Network (JIN).
  - Leverage State investment and reduce risk by building upon the technology and business foundations established by the JIN-sponsored Proof of Concept initiative(s).

- o Leverage State infrastructure. Ensure the agencies involved in the implementation are connected to the most "capable" (secured, bandwidth-capable, reliable) state infrastructure components.

- ***Transfer Implementation Skills and Mentor State Resources.*** Establish a curriculum of learning thru classroomstyle exercise-driven workshops that will develop State skills capability and practically apply these skills towards the development of future message exchange initiatives.
- ***Establish Centralized Operational Support / Service Provision.*** We propose the need to develop the ***JIN Center of Operational Excellence***, an organizational structure and the related programs, policies and procedures, and tools by which the JIN will fulfill its responsibility of operational management to its constituents.

  The diagram below provides a starting point for a collaborative discussion related to the establishment of the JIN's CoE.



The JIN CoE will provide State staff and projects with the tools and repository management that is needed to deliver the projects in support of court transition initiative. The diagram depicts the foundational categories of the CoE by which these tools can be made available to the JIN constituents and partners.

In addition to the development of the JIN Center of Excellence, we would anticipate the need to establish a ***JIN support and maintenance model***. This might involve a technical support and "help desk" capability, the provisioning of minimally disruptive change management services, the management of maintenance software upgrades as well as service component fixes and enhancements, and finally the provision of training offerings for JIN technical staff on the solution components and ISB operations.

It is important to note the difference between the roles of JIN as a service provider (intellectual property management, communications and advisement) versus JIN as an organization of governance with decision-making authority at the agency level. The approach is to allow autonomy for JIN constituents to pursue initiatives within and between agencies. JIN provides AWARENESS and VISIBILITY to the overall integrated JIN and provides the SUPPORT of a centrally maintained and evolving repository of standards, best practices, models and other enablers based on JIN projects.

The Proof of Concept was highly limited in both functional and project approach scope on a number of fronts. While not an exhaustive list, the following points highlight a few of the areas on which little or no project focus or effort was placed – but are areas that would require a much greater level of attention in a larger scale production deployment of an integrated JIN.

- ***Solution Architecture – Definition and Modeling.*** In short, virtually no project process or deliverables were associated with the definition and modeling of the detailed requirements and design of the PoC solution.

This proved to be an “acceptable” approach for the JIN PoC and allowed the exercise to complete with a minimal involvement of the AOC, DOL and DIS.

However, on a production scale the State and JIN should anticipate a much higher level of collaborative involvement in the articulation of current state, requirements, and solution designs. Common techniques and deliverables would include a Functional Requirement Specification, Use Case Models, Business Process Models, Data Mappings and Dictionaries, and a Non-functional Requirement specification addressing {Quality, Scalability, Availability/Redundancy/Recovery, Flexibility, Performance, etc.}.

Especially important in this area would be a focus on architectural design and security. Identify ALL devices between participating agencies – including firewalls, routers, switches, proxies, VPN’s, use of the state network, etc. Account for these in the design of the architecture and test any assumptions as early as possible. Have all hardware and software available before the project kicks off.

Establish a JIN security document of standards and requirements. When designing an exchange with an external agency, be sure to identify the security requirements and capabilities of both JIN and the external agency. Test the agreed upon solution as soon as possible to uncover any issues. All QA/Test environments should have the full security architecture implemented to help identify performance bottlenecks and incompatibilities between exchanges.

- ***Error and Exception Handling***. The JIN PoC was only a simulation of a message between the agencies mentioned above. As there was no production data being passed and no production systems at either end dependent upon the correctness of the exchange and data – virtually no project attention was paid to the modeling and implementation of the processes required to catch and handle errors and exceptions. Again – within a production environment – this work should be accounted for.

- ***Testing***. As the JIN PoC involved virtually no risk of impact to the production environment – no project attention was paid to overall quality assurance and the testing of the message exchanges. A much higher level of involvement of both the project team and business community should be planned for in this area – the development of a testing approach, the development of test plans, and the testing process itself.

- ***Data Architecture***. No intermediate/common XML data format was established. The existing formats for the AOC and SMC transmissions were virtually identical and as such were messaged “as-is”. In a production integrated JIN environment, we would anticipate a much greater emphasis on data architecture in general and in the establishment of common business formats for the message exchanges supported by the integrated JIN.

- ***ESB / Infrastructure Technical Architecture***. Minimal emphasis was placed on ESB configuration in the context of the overall JIN infrastructure architecture. In one day of implementation, ESB domains were quickly established within very small footprints on minimally powered servers at both the DOL and AOC.

  Even considering that a full-scale production JIN ESB would be incrementally implemented, we would nticipate a much greater emphasis would be placed on the modeling of a best-fit technical ESB architecture.

- ***Source Control.*** Ensure that all artifacts are under source code control and that deployments are always from a checked out copy.

- ***Estimating.*** Assuming that multiple incremental project phases will be defined, previous phase estimates should be used as a baseline for estimating future work. As new work is completed, those estimates should be added into the “sample”. Estimates for new work should differ based on changes in scope, complexity, number of exchanges, and changes to technical architecture.

- ***Resourcing.*** In a development team, no more than 1 developer can be considered junior or beginner. All other should be of intermediate to senior experience with the technologies being used. At least on of the team members should have direct experience with the JIN architecture.

- ***Deployment Process.*** The deployment process is a reproducible process for deploying the solution to any one of the different environments. This process should be automated to ensure that human error cannot influence the success of the process.

- ***Issue reporting and escalation Process.*** This identifies the rules for logging issues identified in the exchanges as well as procedures for escalating the issues based on the severity or risk to the production system.

- ***Maintenance Plan.*** Ongoing maintenance of the integration solution is important to ensure that the solution remains in good operation. It is also important to make sure that the solution remains relevant and up-to-date. For integration systems, a planned maintenance regime is an excellent program.

  In addition to facilitating renewed understanding of the integration solution, the maintenance regime provides opportunities for upgrades, and performance metric gathering. By periodically comparing performance metrics against anticipated metrics at design time, changes to messaging volumes can be noticed before causing problems.

4.2 On Web Services

"How would web services work in this architecture, particularly where Java and .Net clients are involved?"
By "*getting away from the technical divide between Java and .Net*" we assume that you:

- Recognize that implementing message exchange interaction using web services does not eliminate the need for an underlying technology implementation (i.e. Java vs .Net)

- Understand that web services provides an interface layer that enables you not to care about the underlying technology implementation – in other words, some JIN constituents or stakeholders may use .Net while others may use Java to implement services – but the web service interface layer provides an insulating layer from the underlying technology implementation.

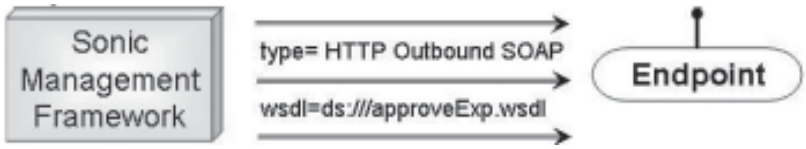
Let us clarify that both the Online implementation team and the ESB technology fully embrace web services – regardless of the technology implementation behind the web service. We see the ESB as being a complementary technology for the JIN direction of a web service enabled technology agnostic implementation as you describe it. The fact that we have implement web services and other integration components primarily in Java is simply a reflection of a localized solution to a specific problem – in short there was no specific requirement to implement as web services.

Web Services technology is a core competence and a primary product direction for Sonic. Sonic's reputation in the WS standards community is well recognized:

- Direct participation: AXIS, WS-Reliability (co-author), WS-ReliableMessaging, WS-Notification (co-author), UDDI
- Actively planning for WS-security, SAML, BPEL
- Early implementer: XML, WSDL, SOAP
- Evangelist & Pioneer: JMS, Web Services, ESB

There are two parts to integrating web services in Sonic ESB. First, the SOAP endpoints are configured to represent the inbound or outbound SOAP connections. This allows any service, itinerary or stateful process to present a Web Services interface, and to orchestrate with any external web services.

The second aspect of web services configuration is the take-on and publication of WSDL definitions. For external web services, you simply attach the WSDL URL to the endpoint definition for accessing the service. For internal services and processes that you are exposing as WSDL's to the outside world, Sonic Integration Workbench assists in generation of the appropriate syntax. The ability to bind SOAP endpoints without writing code, illustrated in the figure below, is another way Sonic ESB promotes loose coupling.



We believe that an SOA integration environment is more than just collection of individual web services. We believe the value of the ESB provides the ability to build an SOA out of an inventory of web-services with features such as orchestrated workflow, business rule processing, error management, auditing, and importantly security. And it provides these "packaged" services in a manner that is agnostic of the technology used to implement the web service itself.

> For example, the City of Seattle and King County develop web services to retrieve offender profiles from their respective information repositories. King County implements underlying web service components with .Net hosted in a BizTalk environment and City of Seattle with Java hosted in an ESB environment. In establishing an SOA for JIN, consideration should be given to binding these services together as a process or orchestrated workflow of processes (e.g. create a single consolidated summary offender profile) but consideration also for a common and re-useable approach to error handling, audit-ability, security over a potentially less than secure network, etc.

Finally, we believe that the ESB itself is the neutral technology you are referring to that provides the messaging architecture support not addressed by web services.

4.3 On Publish-and-Subscribe

"Can a message sent by one producer can be consumed by multiple consumers in this architecture?"

The assumption is correct. ESB uses JMS as the messaging layer that supports both point-to-point and publish-andsubscribe models of distribution. Publish-and-subscribe is supported where a single producer can produce a single message consumed by multiple end-points.

In the context of the JIN PoC implementation – messages produced by the AOC were sent to a topic and were in turn consumed by three subscribing endpoints 1) DOL dB persistence consumer, 2) DOL .Net UI consumer, and 3) a Java applet UI that the team used to show your team the XML messages as they were produced at the AOC endpoint.

4.4 On GJXDM

"Are you seeing any problems with the GJXDM as a data model? If so, can you comment on these problems and proposed solution?"

Based in part on our experiences in the SeaJIS implementation and also on previous work with XML based messaging and integration, we see the following points as the larger issues around the adoption and use of JXDM:

Managing JXDM Recursion:
This refers to the depth or number of levels of the XML hierarchy both through the definition of the element and type hierarchy and the recursive reference o objects within the same object. This creates overhead in the message, and can create computational and transport performance overhead.

- Issues of network performance related to the transmission of unnecessarily large messages are obvious.
- The computational overhead is seen through the depth of the XML structure itself. Due to the hierarchical nature of XML, as the structure of a particular document deepens (i.e. more nodes and branches off of nodes of the tree) the overhead to process the document becomes more computationally expensive as the nodes of the tree must be traversed in order to extract information. If information spans multiple elements, the task of extracting data is also complex when there are values to test for. For example, a person has many

addresses. There are home, work, addresses of friends, family, etc. To find the correct address, we must navigate through all addresses until we find the correct one.

We would recommend, that as part of breaking up the schema into smaller pieces, every effort is made to flatten out the model where is makes sense. For example, in the JXDM, the Street Address is part of another complex type that resides within the Address Type. Although this is architecturally correct, it is not necessarily the most efficient representation as the same grouping could be achieved through naming standards and the additional node in the hierarchy eliminated.

Establishing Usage Guidelines:
Schema Validation versus Schema Interoperability:

Successful validation against the JXDM does not guarantee inter-operability. Because we assume that agencies will extend the JXDM model and because of the use of generic elements in the SuperTypes for example, there are still many opportunities for agencies to create JXDM implementations that are not inter-operable.

Regarding extensions, we assume that the JXDM model must provide for extensions that allow organizations to incorporate attributes that are unique to them within the enterprise. In an example where a Law Department code their prosecuting teams by function and the schema is extended to share this information, there is no guarantee of interoperability with another agency, unless there usage guidelines around how this extension is implemented and used.

Regarding SuperTypes and generic field names, we mean the ancestor types such as ActivityType and SuperType that most other types extend as part of JXDM. The use of SuperTypes with generic field names can create ambiguous attribute names that defeat the goal of using XML to be a “self-documenting” standard.

- For example, there is a CaseType that extends ActivityType. Most Cases have at least one unique identifier – a ‘Case ID’. However, with JDXM, there is no CaseID for the CaseType. The model forces you to traverse up to the ActivityID. As you traverse deeper with the subtypes, it becomes less and less clear which “inherited” field represents what data exactly.

- For example, every single Type inherits all the attributes associated with the SuperType. There are three very similar sounding attributes that can easily demonstrate the aforementioned ambiguity – confidenceNumeric, probabilityNumeric and reliabilityNumeric. All three have very similar definitions. When you consider that multiple agencies are using the GJXDM, and the fields seem fairly open to interpretation, it becomes pretty plausible that different agencies can easily be using different attributes on the model, but really meaning the same thing all along.

We would recommend the establishment of a specification around usage guidelines for making extensions and for using existing elements/attributes in order to improve the inter-operability between implementations.

Another recommendation would be to create more specific types where the element names can more accurately reflect their purpose and remove the generic elements.

- For example, in the ActivityType Type, the ActivityID element currently has a definition of “An identifier that uniquely refers to an activity or process that occurred.”

  There could be cases where agencies use different ID’s to uniquely identify a Case and it would be useful to have a more descriptive element name such as “Case Number” or “Incident Report Number” to support a higher level of interoperability.

Our Approach – Data Architecture
In order to mitigate the risk related to an evolving standard and of using subset schemas, we are implementing a pattern of isolating all end point systems/agencies from the underlying JXDM implementation that is being used (i.e. Loose Coupling).

- From the producing endpoint, we implement a transformation service that takes the XML message from the originating system and transforms it into the JXDM representation (or canonical data representation) before distributing it to the subscribers / participating agencies.

- At the consuming endpoint, we would recommend [and implement where the consuming endpoint is in our domain] the implementation of another transformation service for each participating agency to create a message that they process.

While this pattern allows us to isolate the issues related to the JXDM/canonical data format from the end-point systems but does not eliminate the issues related to JXDM.

GTRI, SEARCH, and GJXDM v3
Online Business Systems recently attended a public training workshop on GJXDM version 3.0 offered by the DOJ and heard both the GTRI and JIEMS approaches to GJXDM implementation.

GTRI acknowledges the difficulties with GJXDM – especially in terms of its large size. They are eager to make life easier for developers by making the data model more user-friendly. They are quite adamant that the best way to go about this is to create schema subsets.

To facilitate this task, a few web-based tools are offered.

GTRI offers two web-based tools that should make life much more bearable, in terms of navigating the GJXDM and paring it down to a manageable schema subset.

- Their first tool is the Date Model Viewer … much like any javadoc setup, you can easily search for any element, and you can see all its properties, its definition, its super-class and anything sub-classes … all hyperlinked. GTRI strongly recommends a combination of using this tool and the actual JXDM spreadsheet, depending on your needs for navigation.

- GTRI also offers a Schema Subset Generation Tool – the idea being that you can build a schema subset on the fly, listing the elements you know you need, and the tool automatically including all the required associated relations (e.g. including the super-classes). The tool lets you build a “wish list” of the types / elements that you know you need … and when you are ready to produce the final schema subset, it will generate a zip file for you … with or without full documentation, depending on the options you’ve selected. This tool is still in the beta stage of development.

- SEARCH offers the Justice Information Exchange Model (JIEM) … this seems to be mostly for document sharing, but at the document element level of things, JIEM ties into the GJXDM, essentially listing every element found in the model. There are plans for enhancement (most likely incorporating what GTRI has already done), to enable someone to easily search the GJXDM and to be able to create a schema subset using a point-and-click interface.

Schema subsets will and do reduce the size of the data model immensely and the tools described above can offer a mechanism to efficiently create these schema subsets. However, the underlying implication is that any developer needs to be highly familiar with the JXDM in the first place. It seems to us that the real bottleneck is the learning curve of the data model in the first place.

Another approach that GTRI strongly recommends along with **conformance validation** is the development of **constraint schemas**.

- The idea behind conformance validation is your schema is a true subset of the GJXDM, therefore if your schema validates then your data is also validated for the entire GJXDM.
- GTRI recommends that developers should also be creating CONSTRAINT schemas. These schemas do not validate the data against the GJXDM at all, but instead are setup up to validate against your local requirements. For example, if you have a Citation, you want to ensure that First and Last name are required, and that there is exactly one Reporting Officer.